Bachelor's thesis

Information Technology

2019

Joonas Aaltonen

AUTOMATIC DEPLOYMENT OF UI TESTING INFRASTRUCTURE USING MODERN DEV OPS TOOLS

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

2019 | 25 pages

Joonas Aaltonen

AUTOMATIC DEPLOYMENT OF UI TESTING INFRASTRUCTURE USING MODERN DEV OPS TOOLS

This thesis discusses automating the deployment of infrastructure required for user interface testing of CGI's case management product. The original goal of the thesis was to enable automatic deployment of servers and software so that the UI of the case management software could be tested without possible issues related to the environment.

The infrastructure used in the thesis project are Windows servers hosted in CGI DevLabs CaaS service. Windows PowerShell scripting language was mainly used as the tool in the thesis, since it allows using all Windows features through the command prompt. In addition to PowerShell, the usability of DevOps tools such as Octopus Deploy and Ansible were studied for the automating purposes.

Octopus Deploy allows packaging built code into installation packages directly from a source control system such as Gitlab, and installing these packages into servers which have the Octopus Tentacle application. Octopus also makes it possible to use variables in configuration files, which allows using only one configuration file which is modified with customer specific values during the deployment process.

An important part of the case management software is OpenText eDOCS document management software. Automating the installation of eDOCS and creating the necessary Windows and SQL credentials with PowerShell scripts is part of this thesis.

The results of the project was multiple useful scripts that can be used for speeding up the installation process of the case management software but the original goal of the thesis was not met.

KEYWORDS:

Case management, Document management, Dev Ops, Software

OPINNÄYTETYÖ (AMK / YAMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2019 | 25 sivua

Joonas Aaltonen

KÄYTTÖLIITTYMÄN TESTAAMISEEN TARKOITETUN SOVELLUSINFRASTRUKTUURIN PYSTYTTÄMINEN MODERNIEN DEVOPS-TYÖKALUJEN AVULLA

Tämä opinnäytetyö käsittelee CGI:n asianhallintatuotteen käyttöliittymän testaamiseen tarkoitetun sovellusinfrastruktuurin pystyttämisen automatisointia. Alkuperäinen tavoite työlle oli mahdollistaa uusien palvelimien nostaminen ja sovellusten automaattinen asennus niin, että asianhallintasovellus olisi valmis ympäristöistä riippumatonta käyttöliittymätestausta varten.

Työhön liittyvät sovellukset toimivat Windows-palvelimella joka toimii CGI DevLabs CaaS -palvelussa. Koska sovellukset toimivat Windows-ympäristössä, työssä käytettiin pääosin PowerShell ohjelmointikieltä, joka mahdollistaa kaikkien Windowsin ominaisuuksien käytön komentorivin läpi. Powershell skriptien lisäksi työssä tutkittiin eri devops-työkalujen, kuten Octopus Deployn sekä Ansiblen hyötyjä ja käyttötarkoituksia.

Octopus Deploy mahdollistaa käännetyn koodin paketoimisen asennuspaketeiksi suoraan versionhallintajärjestelmästä kuten Gitlabista, ja näiden pakettien asennusten suorituksen palvelimille joille on asennettu Octopus Tentacle -ohjelma. Octopus mahdollistaa myös muuttujien käytön esimerkiksi eri asiakkaiden konfiguraatioissa, jolloin yksi konfiguraatiotiedosto versionhallinnassa riittää, ja se muutetaan asiakaskohtaiseksi asennuksen aikana.

Oleellisena osana asianhallintatuotetta on OpenText eDOCS -asiakirjanhallintasovellus jonka asennuksen automatisaation tärkeimpiä vaiheita oli luoda tarvittavat Windows-, ja SQL-käyttäjätilit powershell skripteillä.

Työn tuloksena oli useita käyttökelpoisia skriptejä joita voidaan hyödyntää myös muissa asianhallintasovelluksen asennuksissa, vaikka alkuperäistä tavoitetta ei aivan saavutettu.

ASIASANAT:

Asianhallinta, asiakirjahallinta, Dev Ops, Tietokoneohjelmat

CONTENTS

FIGURES

TABLES

LIST OF ABBREVIATIONS AND TERMS

Deployment	The process of installing and configuring a new software package for use in the target server.
Devops / Dev-ops	Development operations: Methods, processes and tools that are used to make the development process unified, easier and faster.
Development lifecycle	The lifecycle is the order in which the deployments are done into different environments, usually Development, Test and Production. The preferred method is to automatically deploy new versions and nightly builds into the development environment. Once the changes are finished, the test version will be deployed and once it has been tested, the version is deployed into production. When any changes need to be made, the lifecycle is restarted for the new version.
DSC	Desired State Configuration. Powershell management platform that allows the user to enter a desired state of configuration items and the platform executes things to reach the state.
IIS	Internet Information Services. A web service hosting tool available in Windows Servers.
Nightly build	Automatically compiled version of the software that uses the newest available source code. The builds are usually run every night, hence the name.
Production	A version of the software in production is the latest version that the customer is using. No changes are made into this version and all change sets go through the lifecycle before production deployment.
Release	A new built package of software that can be installed.
Smoke testing	The first simple tests to see that all functions work as intended and to see that nothing obvious was broken by the latest changes.
TFS	Team Foundation Server, a source control solution by Microsoft.

1 INTRODUCTION

This thesis is written for a CGI software development team where the author is working in. The ultimate goal of this project is to create a base for automatically running UI tests on the case management software the CGI team is developing. To ensure the credibility of the test results and to limit the possibility of running into environment specific problems, all tests must be run with the same starting conditions. This means that the whole testing environment, i.e. the machine where all software runs must be reset to the same known state after each testing run. To achieve this, the options are to deploy a brand new Windows server for each test run, or to try clearing the installed programs and all possible caches of the server after the tests are done. The first iteration of this project used the latter method due to constrains in the private cloud where the servers are located.

Later discussion with the development team led to a decision to try to make the automatic deployment process separated from the testing to make it possible to utilize that in later installations of the case management software.

The simplified description of this project is to identify all steps in setting up a server for the case management software and after that, to try to automate all those steps with powershell scripts. Once the necessary scripts are working, they will be started remotely, likely with Octopus Deploy, to complete the prerequisite tasks. Octopus will deploy all the necessary software on the server after which the tests can be run. After the tests, Octopus will be used to remove all the installed software and to revert the server to the same state as before the deployments.

During this thesis project, a senior devops architect started working in the team. He provided a lot of knowledge about advanced devops tools and helped to guide this project into a better direction but this rendered some of the early work done useless and required to change or update multiple ideas.

2 REQUIREMENTS AND SPECIFICATIONS

This thesis was written for development process of an enterprise case management software that has integrations to other programs and can be customized for each customer to suit their specific needs. These customizations are often functionalities related to the customer's own systems or integrations with other applications to support the workflows. This means the user experience may be different for different customers but the application deployment and installation process on the server does not change meaningfully. The case management software can have either *Microsoft SharePoint* or *OpenText eDocs DM* as its document handling component depending on the customer's preference, but this thesis will only cover the eDocs DM integration. The software uses a variety of service accounts that run the background services. Some of these accounts are domain users while some are local users created manually for each installation. This means automatizing the user creation will also benefit future installation processes.

The databases for all software are instances of Microsoft SQL Server 2016. To ensure the same initial state, all databases and database logins must be removed after tests and restored or created for each test run.

The development team mainly uses a private cloud by CGI DevLabs for development and this cloud environment allows creation of a new empty server for this project. In theory, it should be possible to automatically create customized virtual machines into the cloud, but that requires higher level access to the management tools than the one available, so the project will use only one server that is manually created.

The software is hosted on Windows 2016 Servers and each customer has a copy of their specific version of the software on their own server. Each server hosts up to 5 instances of the same software, these being the different "environments", development, feature, QA, test and sometimes production. All new changes are pushed into the development environment where they are tested. After the smoke testing, they can be deployed into the QA environment where they can be thoroughly tested later in regression tests. Once the next version update for the customer is ready to be brought to customers test environment, it is also deployed to CGI test environment to make sure everything related to the installation process is working. Usually the version in the CGI test environment is the same as in customer's production but occasionally there can be a production environment which holds the copy of the current customer production version, in case a

new version is deployed to customer's test but will not be installed to production. The feature environment allows deployment and testing of new features that can break existing functions.

The case management software is hosted in Windows IIS and each of the environments have their own virtual directory. The user can access any of the environments with a browser by navigating to https://<servername>/**<environment>**/<service>

3 AVAILABLE TOOLS AND TECHNOLOGIES

This chapter discusses the tools and technologies available for completing this project. In theory, the only limitation for tools to be used was that they need to be licenced to be used commercially for free. CGI has a strict policy on using trial versions of software so that kind of solutions are not available. Another factor is that the development team has a set of tools that are currently used, so it would be desirable to utilize these tools as much as possible in this project to avoid generating extra complexity if the automation is used in deployments. In practice, this means that the automatized process should use the same tools as are currently used for the manual deployments.

Powershell scripting allows executing almost any task in Windows environment programmatically and also has the basic .Net capabilities, so writing powershell scripts for various tasks ranging from Windows user creation to SQL-server operations is possible. This approach allows a very modular way of constructing the project as each operation can be separated into its own script which then can be called from another script when needed. Most of the automation tools introduced below also use Powershell to run their processes. The main benefit in using them is that these tools can provide some template code for various tasks, so the user does not need to start from scratch with simple things like adding user accounts. The automation tools also run the scripts in a managed way so it is not necessary to write the separate “backbone” script that calls each task or have the scripts call the next one, which means the execution chain may stop if just one script fails.

3.1 Octopus Deploy

Octopus Deploy is a tool that allows deploying your builds from the source control such as Microsoft Team Foundation Server or Gitlab into the target servers. It allows monitoring and controlling the releases with rules, for example requiring a deployment into development and test environments before creating the production release.

The deployments are done either automatically through an endpoint called “Octopus Tentacle” or as an offline package drop that has all scripts included, but needs to be manually run in the target server. The tentacle allows the Octopus server to communicate with the target server, transfer the installation files and execute them there. Once the

Windows environment is set up, Octopus can be used to deploy the case management software into the test server.

The tentacles can be setup in two different ways: a listening tentacle and a polling tentacle. The listening tentacle listens to the configured TCP port in target server and waits for the invocation from the deployment server. The polling tentacle sends checks to the deployment server periodically to see if there is a task waiting for execution. Octopus suggests using listening tentacles and all tentacles in the CGI environment have been configured to the listening mode. (Octopus Deploy, 2019)

3.1.1 Octopus Project variable handling

One feature in Octopus is the ability to create deployment variables into the projects, allowing the same codes to be deployed with different configurations. The variable names are written into the program configuration files and the values are stored in the Octopus server where they can be modified through the web user interface. During the deployment, the Octopus scripts change the variables in the configuration files into the right values to be used in the target server.

Following an example of using Octopus variables in XML configuration file:

The original syntax for the configuration file has the Octopus variable:

```
<Application-Settings>
<Target-Server name=”#{ServerAddress}”/>
</Application-Settings>
```

The Octopus user interface has the variable, its value and scope if different values are used in separate environments. Example of this in Table 1.

Table 1 – Octopus variable examples 1

Variable Name	**Value**	**Scope**
ServerAddress	http://myserver.dev.local/dev	Development, Test
	http://myserver.dev.local/prod	Production

If the deployment is done into development environment, the configuration file has the following field and value:

```
<Application-Settings>
<Target-Server name=”http://myserver.dev.local/dev”/>
</Application-Settings>
```

The variable values can also include other variables and it is possible to chain multiple variables, which allows for example, to construct a dynamic URL from multiple variables as done in Table 2:

Table 2 – Octopus variable examples 2

Variable name	**Value**	**Scope**
Binding	http://	Development
	https://	Production
ServerAddress	myserver.dev.local	Development, Production
ServerEnvironment	/dev	Development
	/prod	Production
FullAddress	#{Binding}#{ServerAddress}#{ServerEnvironment}	Development, Production

Deploying the variable *FullAddress* into *Production* environment would result into the following URL: https://myserver.dev.local/prod

3.2 Gitlab Continuous Integration and Delivery (CI/CD)

The CGI team uses Gitlab for source control alongside Microsoft TFS. All projects have master and development code branches where the continuous integration is active, meaning every merge into the branches starts the unit test runner and build process that creates the deployable package for Octopus Deploy if the build succeedes. Development branch is also used for nightly builds that are automatically deployed to test servers with the Octopus Tentacle.

Once the project for automating the environment setup is done, the nightly builds would be deployed into the test server where UI tests could be run.

3.3 CGI Devlabs private cloud (CaaS)

All operations will be conducted in the private cloud where the development of the product to be tested is done. All servers and workstations are in the same domain and the users generally login to the machines through remote desktop access, using the domain credentials. Both domain and local credentials can be created if required in this project.

The private cloud has multiple CGI development teams as customers, and each customer has their own Containers as a Service (CaaS) environment which allows freely creating servers into the cloud within the resource limits.

3.4 Ansible and Chocolatey

Ansible is an automation engine that allows running remote operations with Powershell into computers in your network. A local Ansible admin user needs to be created in the computers that then runs the Ansible scripts with required privileges. To better manage the Ansible projects, an open-source management tool called AWX is being used in the development domain.

A user can create different sets of scripts called “Roles” in Ansible which then can be deployed. For this project, the role will include installation scripts for the necessary programs but Ansible is also used for deploying development workstations. The workstation role includes scripts for installing development tools such as Visual Studio and different software development kits.

Chocolatey is a package manager for windows that allows downloading and installing of over 6000 different programs (Chocolatey #1, 2019). It can be used in the same manner as “apt-get” in Linux environments, running commands from powershell to install any hosted package, but in addition Chocolatey also has a GUI to help users that are not as proficient with powershell (Chocolatey #2, 2019). Chocolatey will be used for installing or checking the existence of required software on the test machine.

If a large number of software needs to be downloaded and installed, the combination of Ansible and Chocolatey would allow the user to command Chocolatey through the Ansible scripts. First installing Chocolatey from a local installation file and then downloading and installing programs with the Chocolatey commands. Ansible and Chocolatey were mainly used by the team's DevOps architect during the project and thus this thesis doesn't extensively focus on these tools.

4 ALTERNATIVE SOLUTIONS

4.1 Microsoft Azure

The team has a small number of virtual machines running in Microsoft Azure but all the development workstations and the customer specific case management servers are hosted in the CGI private cloud. The main reason for this is the pricing and the level of support needed for the servers. The private cloud contains over 90 machines but not all of them are in active use. Even if with Azure the price is charged on “pay what you use” basis, hosting this number of machines would cost several times more.

Since almost everything else is done in the CGI private cloud, it was decided that the tests should be conducted there too. Azure has its own devops toolkit that might have been useful for the project, but after the devops architect joining the team, the devops capabilities as well as the number of tools in the team have been massively increased.

4.2 Containers

Containers are a method of virtualization. Unlike Virtual Machines, containers virtualize the software but not the hardware. This means the containers share a host operating system which handles the access to the computer’s hardware resources, while Virtual Machines have the Hypervisor acting as the hardware layer while it is actually running on the host OS and only relaying the hardware calls. The difference of these methods is illustrated in the figure 1 (Connolly, 2018).

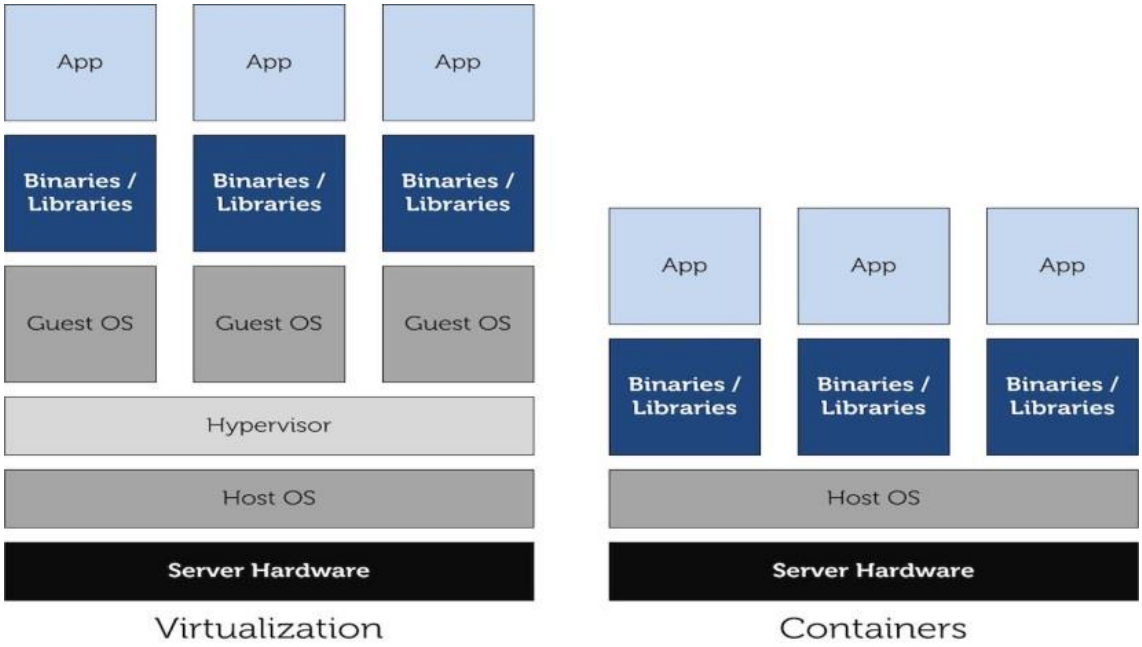

Figure 1. Container stack compared to virtual machine stack.

When using containers for software hosting, the software and all its dependencies are packaged into the container image which then is hosted with a container engine such as Docker. (Docker, 2019) This allows moving the containerized application into different hardware or operating system as the application side only communicates with the virtualized guest OS. This feature can be used for rapidly deploying software, essentially doing the same process as this thesis project on one button press.

Containerizing the product is a long term goal for the team but currently it does not perfectly support it so we are not using containers for deployment.

5 IMPLEMENTATION

5.1 Windows server preparations and feature installations

The software requires a number of Windows server roles and features to be installed in order to run. As mentioned earlier, it is hosted on Windows IIS which is one of these Windows Server OS roles. These roles and features are built in to the operating system but need to be enabled for use. Enabling can be done manually through the *Server Manager* tool where all features are shown on select lists, or automatically with Powershell. The Powershell command for installing the Windows features is:

```
Install-WindowsFeature
[-Name]
[-IncludeAllSubFeature]
[-ComputerName <String>]
```

(Microsoft Docs #1, 2019)

This means that to install the features, one only needs to know the computer name where the features are to be installed and list the correct feature names to iterate through that list with this command. The second parameter, include all sub-features can be used if a role is installed with all of its features. In script 1, one can see that the first array of features is installed one feature at the time but the second array which contains .Net Frameworks and other features is installed using the include all sub-features flag.

SCRIPT 1 – Windows features

```
# Script to install Windows server features offline
# Run on the computer where you want the features installed

# Get the computer name where the script is running
$computerName = $env:COMPUTERNAME

$featureNames =  @( # IIS features
                    "Web-Server","Web-WebServer","Web-Common-Http",
# < ---- A massive number of other IIS features, omitted to save space ---- >
                    "Web-Http-Errors", "Web-Includes",
                    # Misc tools
                    "Web-Mgmt-Console","Web-Mgmt-Service", "Storage-Services")

$otherFeatures = @( # .NET frameworks and Other features
                    "NET-Framework-Features","NET-Framework-45-
Features","WAS","PowerShellRoot","WoW64-Support","MSMQ-Server")

# Install windows features
foreach ($feature in $featureNames)
{
    Install-WindowsFeature -Name $feature -computerName $computerName
}

# Install the ones that will have all subfeatures
foreach ($feature in $otherFeatures)
```

```
{
    Install-WindowsFeature -Name $feature -IncludeAllSubFeature -computerName
$computerName
}
```

Once IIS is installed, a default web page is created where the web services will be hosted. This default website requires minor tweaking such as assigning HTTPS bindings and a certificate.

5.1.1 IIS Default web site configuration

A script using the Windows *Desired State Configuration* or DSC, (Microsoft Docs #2, 2019) was created by the team's DevOps architect to handle the whole IIS setup and default website configuration. In Script 2, it is the part of the DSC script that handles the default website.

```
SCRIPT 2 – Default website
xWebsite DefaultSite
        {
            Ensure          = "Present"
            Name            = "Default Web Site"
            State           = "Started"
            PhysicalPath    = "C:\inetpub\wwwroot"
            BindingInfo     = @( MSFT_xWebBindingInformation
                               {
                                   Protocol                = "https"
                                   Port                    = 443
                                   CertificateThumbprint   = $CTB
                                   CertificateStoreName    =  'My'
                                   IPAddress               = '*'
                                   SSLFlags                = '0'
                               }
                               MSFT_xWebBindingInformation
                               {
                                   Protocol                = "http"
                                   Port                    = 80
                                   IPAddress               =  '*'
                               }
                               MSFT_xWebBindingInformation
                               {
                                   Protocol                = "net.pipe"
                                   BindingInformation = '*'
                               }
                           )

        }
```

The array of binding informations is the part that would be done by hand if a script wasn't run. The script assigns bindings for https, http and net.pipe protocols. Going through the script from bottom up, the net.pipe binding is used for different services to communicate with eachother in the same machine. Thus it doesn't require a port or IP address. Http binding has the default port 80 assigned and it listens to all unassigned IP addresses.

Https binding is slightly more difficult since securing the connection requires a certificate. This certificate should be installed in the Windows personal certificate storage and the thumbprint of the certificate should be given to the script on execution (used in the variable $CTB). The assigned 443 port is the default for https.

5.2 SQL-Server preparations

5.2.1 Renaming the server instance

The Windows servers where everything is hosted are deployed from a template that has Microsoft SQL Server 2016 installed. The name of the server is changed from “Template” to customer’s name after Windows installation but the SQL server instance name needs to be changed manually with an SQL command.

The final version of the script will be written into a powershell script that is run by Ansible but the core idea of the command is the following:

```
(SCRIPT 3 – SQL Server naming)
DECLARE @NewServerName varchar(100) = ‘ServerNameFromScriptParameter’
DECLARE @OldServerName varchar(100) = @@SERVERNAME

Print 'Current server name ' + @OldServerName + ' | New server name ' +
@NewServerName

EXEC sp_dropserver @OldServerName
EXEC sp_addserver @NewServerName, 'local'

-- To apply the change, restart the SQL instance!
```

The name for the new server is entered to Ansible when the deployment scripts are run and the parameter is passed to the SQL command through powershell. The existing servername is saved to the variable with the @@SERVERNAME command. The print statement will be redundant in the final version of the deployment process but all debugging information is valuable in the early stages. Instead of printing into the console it may be changed to log the information into installation logs or the values are passed back to the Ansible powershell script that allows the user to see them. The execution statements then drop the old server name and add the new name. The syntax for the sp_addserver command requires the parameter *‘local’* when the new server name is

added to the current instance. To apply the change, the SQL server needs to be restarted which can be done with another powershell command or by restarting the whole Windows server during the deployment process.

5.2.2 Creating and restoring the required databases

Generally each of the customer specific Windows server hosts the 4 different instances of the same case management software. Each environment requires its own set of databases. These databases are often restored from existing database backups instead of creating fresh databases, but in case of a new customer, a new set of databases needs to be created.

Microsoft SQL Studio management tool doesn't allow restoring multiple databases with a single selection so restoring tens of databases is tedious. A simple .Net console application was created for this purpose. The application creates a connection to the SQL server, asks the user for a directory path for the database files and runs the SQL Restore commands for each of the databases in a specified folder. Since some of the databases have been created years ago and have gone through migrations from older SQL server versions and older product versions, the restoring operation can sometimes crash for various reasons. The databases will also be restored with the same name as they were backupped so if needed, renaming the databases needs to be done separately. These issues mean running the database restore should be done by hand instead of in a deployment script. A script for creating a brand new set of databases will be written for the final version of the deployment process but is not yet done.

5.3 User management and creation

To setup the environment, two service accounts need to be setup as local administrators. These accounts are domain users so they are queried from the active directory and added into Windows' local admin user group. These two service accounts run the Windows services for eDocs DM and the CGI product. Development is currently in the works for integrating a reporting service running Qlik Search into our product. Once that is ready, a third service account will be required for the Qlik Search service.

Microsoft development blog provided a good example (Script 3) for adding an existing domain account into the local admin group with powershell.

```
SCRIPT 4 - Add domain user to local users
$objUser = [ADSI]("WinNT://dev/DMService")
$objGroup = [ADSI]("WinNT://Aaltonen-DevOps/Administrators")

$objGroup.PSBase.Invoke("Add", $objUser.PSBase.Path)
```

This script is a good showcase of how efficient Powershell can be in professional hands. The [ADSI] type adapter creates a reference to the active directory which is where the domain users are stored. The WinNT provider is used for working with local users and groups but it doesn't recognize the active directory components, so the syntax for domain user is *domain-name/username*. This can be confusing as generally in Windows, the domain users are called with *domain-name\username* syntax where a backslash is used instead. The local user group uses the same syntax, in this case *computer-name/group-name* but it could also be *domain/user-group-name* if the intention was to add user into a domain user group. On the last row the PSBase object is used to access the invoke method of the objGroup variable which allows adding the user into the group (Microsoft Devblogs, 2008) .

The ADSI active directory component allow this script to work remotely when the computers are in the same domain and in the same AD. If in the script the Aaltonen-DevOps computer name is changed into Aaltonen-Dev, which is another available machine, it will add the DMService user into that computer's local admin group, even when running this script on the DevOps machine.

Two local accounts are created to the SQL server. One is the admin account for the DM service that executes its database operations and the other is database migrator that handles database operations during the Octopus deployment. Since we're using Microsoft SQL server, a couple of options were available to create the SQL accounts; to use Powershell's SQL commands to directly create the accounts or to open a connection to the server and execute the commands in SQL inside the server.

The main functions of the script were opening the connection to the database and executing the SQL command that creates the database user and logging the process.

```
SCRIPT 5 - Logging snippet
Function LogWrite
{
   Param ([string]$logstring)

   Add-content $Logfile -value $logstring
```

```
}
```
(JNK, 2011)

The logging in script 5 was based on sample code that was posted on StackOverflow. It allows the user to designate a file path to a log file where all logs will be appended. The function works exactly like the output into the Powershell terminal which makes logging very easy.

SCRIPT 6 – Sql commands

```
$SQLStatements = @("USE [master]", "CREATE LOGIN [<UserName>] WITH
PASSWORD=N'<Password>', DEFAULT_DATABASE=[master], CHECK_EXPIRATION=OFF,
CHECK_POLICY=OFF")
```

The script 6 shows an array where the required SQL commands to set database context to master database and to create the user login are. Setting the database context needs to be executed alone, so the commands are stored in an array from where they can be executed separately.

SCRIPT 7 – Opening SQL connection

```
$SQLConnection = New-Object System.Data.SQLClient.SQLConnection
    $SQLConnection.ConnectionString = "Server=Aaltonen-Dev\SQL16;
Trusted_Connection=True"
    $connectionFailed = 0

    try
    {
        $SQLConnection.Open()
        LogWrite ("{0} -- Connection opened to SQL Server`r`n"
 -f (Get-Date).ToString("HH:mm:ss"))
    }
```

The script 7 includes the part of the script that opens the connection to the SQL server. The database connection uses trusted connection method which means the account that runs the script is also used to connect to the SQL server, and thus no user credentials are required.

SCRIPT 8 – Executing SQL commands

```
$SQLCommand = $SQLConnection.CreateCommand()
        $SQLCommand.CommandTimeout = 1800

        foreach ($command in $SQLStatements)
        {
            $SQLCommand.CommandText = $command
            LogWrite ("SQL command sent to the server:`r`n{0}`r`n" -f
$SQLCommand.CommandText )

            try
            {
                $SQLCommand.ExecuteNonQuery() | Out-Null
            }
```

Finally in script 8 the array declared in the beginning is looped through and both of the commands are executed into the SQL server. Both commands are also logged to make

sure the syntax is right. Executing the commands and connecting to the server are done inside a try-clause to enable logging of errors in case an exception happens while doing these operations.

5.4 DM Installation

The installation media for eDOCS DM server is provided as Microsoft installer (.msi) file, which makes it fairly straightforward to install it through a command prompt. Microsoft has released a tool called Orca which allows opening the .msi files into easily readable table form which then allows you to see all the possible parameters that can be given in the installation. This allowed me to find the needed parameter names for the license key and its password so they can all be entered in the installation phase instead of reconfiguring later.

Ultimately the DM server installation command was very simple:

```
SCRIPT 9 - DM Server install
msiexec.exe /i "C:\Install media\edocsdm16.2server.msi" /passive
OT_SERIALNUMBER_17=<Serial number>
OT_PASSWORD_17=<Encryption password> /Lv
"C:\Install media\log.log"
```

MSIExec program is the installer that runs .msi files and it can be used for installations, install repairs and uninstalls. The /i flag indicates installation and the path after that is the location of the .msi package to be installed. Due to the strict security monitoring of their private clouds by CGI DevLabs, running the script in quiet mode is not suggested as that can raise alarms in the security center since the same method is often used by malware. With the /passive -flag the installer window will be shown but it runs in passive mode, meaning the user cannot or does not need to interact with it. Verbose logging is saved into the path specified as the last thing. Logging is not necessary, but if something breaks later on, having as much logs as possible is useful for finding the root cause for the issue.

The DevOps architect of the team later created an Ansible role to install all the DM services with the necessary parameters. This includes the DM server and classic and new versions of the web server. The installation uses the msiexec scripts practically identical to the script 9, but they are executed with Ansible role that also makes sure all installation medias and scripts are in the right place before starting.

The DM server requires an .ini-file called PCDOCS.ini which includes the information about the DM libraries used in the server. In the “advanced” version of the script done with Ansible, a template was used to create the file by overwriting the template values with the library parameters supplied by the user.

5.5 Octopus tentacle installation

The Octopus tentacle endpoint installation is done in an Ansible role. The script takes a customer name, the Octopus instance name and a server IP as parameters. This allows the installation to automatically connect it to the Octopus server and to the correct deployment target in Octopus.

6 CONCLUSION

The original goal of this thesis project was to automate the UI testing in the development process, and this goal was not reached. However, the findings during this thesis and time dedicated for the project helped creating an overview of what is required to achieve the goal of fully automatized test environment.

The current situation allows fast deployment of all necessary components of the software into an existing server but the goal of creating temporary servers from scratch for the tests was not reached. The necessary steps to achieve this have been discussed with the devops expert of the team and this will be something to be looked at in the future.

The next steps in utilizing the findings will be educating other developers in the team to use the newly utilized devops tools, such as Ansible to help the process of deploying new servers. The scripts written will be used to create failsafe portable tools for installing the various components to customer environments where it is not possible to use Ansible or other online deployment tools.

REFERENCES

Chocolatey #1, (2019) *Chocolatey* [online] Available at: https://chocolatey.org/ [Accessed 8 May 2019].

Chocolatey #2, (2019) *Chocolatey – About* [online] Available at: https://chocolate.org/about [Accessed 8 May 2019]

Connolly, M. (2018) *Lecture Topic 3 - Infrastructure Models* [Slide show] Presented in Cork Institute of Technology 26 Sep 2018

Docker Inc. (2019). *What is a container* [online] Available at: https://www.docker.com/resources/what-container [Accessed 4 Apr. 2018]

JNK, (2011) *Create log file in powershell* [online] Available at: https://stackoverflow.com/a/7835668/7640804 [Accessed 1 Jul, 2019]

Microsoft Devblogs, (2008). *Hey, Scripting Guy! How Can I Use Windows PowerShell to Add a Domain User to a Local Group?* [online] Available at https://devblogs.microsoft.com/scripting/hey-scripting-guy-how-can-i-use-windows-powershell-to-add-a-domain-user-to-a-local-group/ [Accessed 11 Mar. 2019]

Microsoft Docs #1, (2019). *Install-WindowsFeature* [online] Available at: https://docs.microsoft.com/en-us/powershell/module/servermanager/install-windowsfeature?view=winserver2012r2-ps [Accessed 29 Sep 2019]

Microsoft Docs #2, (2019). *Windows PowerShell Desired State Configuration* [online] Available at: https://docs.microsoft.com/en-us/powershell/dsc/overview/overview [Accessed 29 Sep. 2019]

Octopus Deploy, (2019). *Tentacle Communication Modes - Octopus Deploy*. [online] Available at: https://octopus.com/docs/infrastructure/deployment-targets/windows-targets/tentacle-communication [Accessed 29 Sep. 2019]